Almost Always Auto

Wolfram Rösler

https://gitlab.com/wolframroesler/Talks/blob/master/aaa.pdf

Defining a variable is easy

```
int i;

double d = 0;

std::string s("Hello");

MyType x{1, 2, 3};
```

Or is it?

```
std::map<int,std::vector<std::string>> map;

std::map<int,std::vector<std::string>>::const_i
terator it = map.begin();

int key = it->first;

std::vector<std::string> value = it->second;
```

C++11 auto to the rescue!

```
std::map<int,std::vector<std::string>> map;

auto it = map.begin();

auto key = it->first;

auto value = it->second;

for(auto& it : map) { … }
```

Sometimes auto is required

```
auto square = [](int i) { return i * i; }
```

BTW, auto is not a new keyword

```
/* Storage classes in classical C: */

auto int a;           /* automatic storage */

static int b;         /* static storage */

register int c;       /* register storage */

extern int d;         /* external storage */

int e;                /* auto is default */
```

Suggestion: Use auto all the time

```
int i;                   auto i = 0;

double d = 0;            auto d = 0.0;

std::string s("Hello"); auto s = "Hello"s;

MyType x{1,2,3};         auto x = MyType{1,2,3};
```

Toward correct-by-default, efficient-by-default, and pitfall-free-by-default variable declarations, using "AAA style"... where "triple-A" is both a mnemonic and an evaluation of its value.

– Herb Sutter, https://herbsutter.com/2013/08/12/gotw-94-solution-aaa-style-almost-always-auto/

Benefits of AAA style

- Can‘t forget to initialize
- No repetition of type names
- No implicit conversion or narrowing
- Unified “name first” style
- Simplified refactoring
- Less code to type, less code to read

But I can’t see the data types!

```
auto getPixmap(QUuid uuid)
{
    auto cached = checkCache(uuid);
    if (cached) {
        return *cached;
    }

    auto image = loadImage(uuid);
    auto pixmap = QPixmap::fromImage(image);
    putIntoCache(uuid, pixmap);
    return pixmap;
}
```

And I need to see the data types!

Do you really?

```
template<class Container, class Value>
void appendIfNotYetThere(Container& c, const Value& v)
{
    if (std::find(c.begin(), c.end(), v)==c.end()) {
        c.emplace_back(v);
    }
}
```

Defining a string, then and now

```
/* C 1975 style */
char str[MAXLEN+1];
char *ptr = malloc(MAXLEN+1);


// C++ 1995 style
std::string str;
```